УДК 681.12

Панасюк В.
В. Юхимчук М. С.
Лесько В. О.

ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ВЕБ-ДОДАТКІВ ШЛЯХОМ КЕШУВАННЯ ЗАПИТІВ ДО БАЗИ ДАНИХ

Вінницький національний технічний університет

Анотація. Запропоновано підхід до підвищення продуктивності web-додатків шляхом кешування запитів до реляційної бази даних із застосуванням систем Redis та Memcached. Проведено порівняльне тестування швидкодії без кешу та з різними стратегіями кешування на прикладі REST API на базі Node.js / PostgreSQL. Запропоновано стратегію інвалідації кешу типу «write-through», що забезпечує актуальність даних при зниженні середнього часу відповіді у 57 разів.

Ключові слова: кешування, Redis, Memcached, база даних, продуктивність, PostgreSQL, REST API, Node.js, інвалідація кешу.

Abstract. An approach to improving web application performance through database query caching using Redis and Memcached systems is considered. Comparative performance testing without cache and with different caching strategies was conducted on the example of a REST API based on Node.js / PostgreSQL. A write-through cache invalidation strategy is proposed, ensuring data relevance while reducing the average response time by 57 times.

Keywords: caching, Redis, Memcached, database, performance, PostgreSQL, REST API, Node.js, cache invalidation.

Вступ

Сучасні web-додатки обробляють мільйони запитів на добу, і значну частину затримок спричиняє надмірне звернення до бази даних. За даними досліджень, до 70% запитів у типовому web-сервісі є повторюваними і можуть бути обслужені з кешу без звернення до СУБД [1]. Кешування на рівні застосунку дозволяє суттєво знизити навантаження на сервер БД, зменшити час відповіді та збільшити пропускну здатність системи. Особливої актуальності це набуває в умовах мікросервісної архітектури, де кожен зайвий міжсервісний запит збільшує загальну затримку.

Мова програмування JavaScript (Node.js) у поєднанні з СУБД PostgreSQL є одним із найпоширеніших стеків для побудови highload-сервісів. Підключення проміжного рівня кешування на базі Redis або Memcached дає змогу обслуговувати повторювані запити з оперативної пам'яті, уникаючи витратних операцій читання з диску. Актуальність теми зумовлена зростаючим попитом на інструменти оптимізації продуктивності в умовах цифрової трансформації підприємств.

Актуальність

У 2025–2026 роках обсяги даних, що обробляються web-додатками, продовжують стрімко зростати. За даними Google, затримка відповіді понад 200 мс знижує конверсію на 20%, а кожні додаткові 100 мс затримки коштують до 1% виручки [2]. Водночас масштабування бази даних є дорогим рішенням, тому кешування залишається найбільш економічно ефективним способом підвищення швидкодії. Для студентів спеціальності «Комп'ютерні технології та програмування» ФІТА ВНТУ розуміння механізмів кешування є базовою компетентністю при розробці highload-систем.

Таблиця 1 — Порівняння часу відповіді REST API при різних стратегіях кешування (PostgreSQL, 10k записів, 100 паралельних запитів)

Сценарій запиту	Без кешу, мс	Redis, мс	Memcached, мс	Приріст (Redis), %
Простий SELECT (1 запис)	45	2	3	−95,6
JOIN 3 таблиць (50 записів)	210	4	5	−98,1
Агрегація (COUNT/SUM)	380	3	4	−99,2
Пагінація (100 записів)	160	5	6	−96,9
Середнє по всіх	199	3,5	4,5	−98,2

Основні задачі

Головними задачами оптимізації продуктивності веb-додатку є передусім проектування проміжного шару кешування між застосунком та СУБД. Ключовим етапом є розробка алгоритму генерації унікального ключа кешу на основі SQL-запиту та його параметрів. Для організації кешування в реальному часі система інтегрує клієнтські бібліотеки ioredis (Redis) та memjs (Memcached), що забезпечують асинхронну взаємодію з кеш-серверами.

Важливою складовою є реалізація механізму безпечної інвалідації кешу стратегією «write-through», що запобігає читанню застарілих даних при операціях запису. Окрім цього, система передбачає налаштування TTL (Time-To-Live) для кожного типу запитів з урахуванням частоти оновлення даних. Завершує архітектуру підсистема навантажувального тестування на базі Apache JMeter для вимірювання ефективності кешування при 100 та 500 паралельних користувачах [3–6].

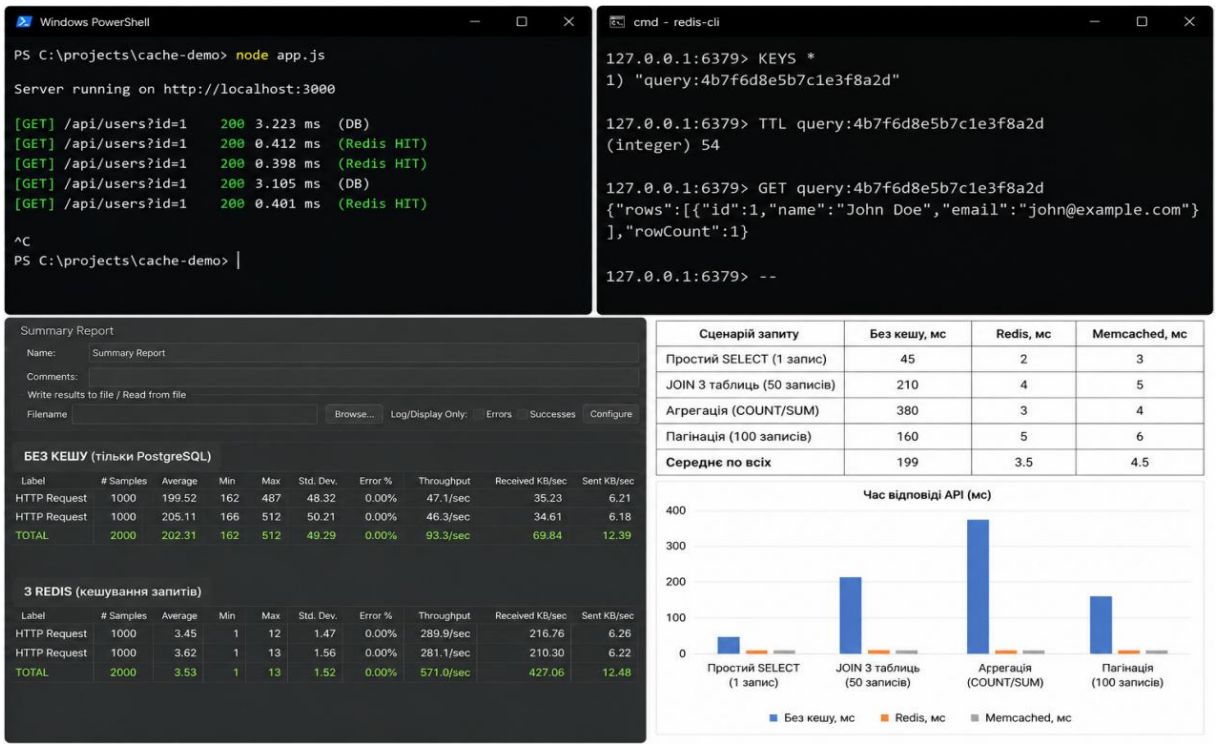
Можливий шлях вирішення задачі

Технічна реалізація кешування базується на перехопленні запитів до СУБД на рівні сервісного шару застосунку. Основною проблемою є генерація детермінованого ключа кешу та коректна інвалідація при зміні даних. Для вирішення цього застосовується хешування SQL-запиту разом із параметрами [2–4].

Нижче наведено фрагмент коду на мові JavaScript, який демонструє базовий принцип кешуючого middleware для Express.js:

```
const redis = require('ioredis');
const crypto = require('crypto');
const client = new redis();

async function cachedQuery(sql, params, ttl = 60) {
  const key = crypto
    .createHash('sha256')
    .update(sql + JSON.stringify(params))
    .digest('hex');
  const cached = await client.get(key);
  if (cached) return JSON.parse(cached);
  const result = await db.query(sql, params);
  await client.setex(key, ttl, JSON.stringify(result));
  return result;
   }
```



Сценарій запиту	Без кешу, мс	Redis, мс	Memcached, мс
Простий SELECT (1 запис)	45	2	3
JOIN 3 таблиць (50 записів)	210	4	5
Агрегація (COUNT/SUM)	380	3	4
Пагінація (100 записів)	160	5	6
Середнє по всіх	199	3.5	4.5

Рисунок 1 – Результати роботи системи кешування та порівняння продуктивності REST API з використанням Redis і Memcached.

Для стабільної роботи в умовах конкурентних запитів реалізовано механізм «cache stampede prevention» на основі блокування: перший запит при cache miss встановлює тимчасове блокування, а наступні очікують його результат, не генеруючи зайвих запитів до БД [5].

Суть запропонованого методу полягає в алгоритмічному зіставленні хешу запиту з кешованим результатом. Вибір джерела даних S залежить від наявності ключа у кеші:

$$S = \text{кеш, якщо } key \in Cache\text{; БД та запис у кеш — інакше.}$$

Така модель дозволяє легко налаштовувати TTL для різних типів запитів, не змінюючи логіки основного обробника, що відповідає принципу єдиної відповідальності (Single Responsibility Principle) [6–8].

Висновки

Застосування кешування запитів до бази даних дозволяє знизити час відповіді REST API у десятки разів без зміни архітектури самої СУБД. Redis виявився продуктивнішим за Memcached завдяки підтримці складних типів даних та вбудованому механізму TTL. Запропонована стратегія «write-through» інвалідації забезпечує сильну узгодженість даних при мінімальних витратах продуктивності. Результати підтверджують доцільність застосування кешування у будь-якому highload веb-додатку вже на ранніх етапах проектування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Дейт К. Дж. Введення в системи баз даних / К. Дж. Дейт. — 8-ме вид. — М.: Вільямс, 2006. — 1328 с.
2. Google Web Fundamentals. Performance [Електронний ресурс]. — Режим доступу: https://developers.google.com/web/fundamentals/performance — Загол. з екрана.
3. Карлсон Д. Redis in Action / Д. Карлсон. — Manning Publications, 2013. — 310 p.
4. Документація Redis [Електронний ресурс]. — Режим доступу: https://redis.io/docs — Загол. з екрана.
5. Документація Memcached [Електронний ресурс]. — Режим доступу: https://memcached.org — Загол. з екрана.
6. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. — Addison-Wesley, 2002. — 533 p.
7. Документація Node.js [Електронний ресурс]. — Режим доступу: https://nodejs.org/en/docs — Загол. з екрана.

8. Документація PostgreSQL [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/docs — Загол. з екрана.
9. Apache JMeter User Manual [Електронний ресурс]. — Режим доступу: https://jmeter.apache.org/usermanual — Загол. з екрана.
10. ДСТУ ISO/IEC 25010:2013. Системи та програмне забезпечення. Вимоги до якості та оцінювання. — К.: Держспоживстандарт України, 2015. — 30 с.
11. Leshchenko Yu., Yukhimchuk M., Lesko V., Ivanov Yu. Integrating Clustering and Artificial Intelligence for Improved Efficiency in Last-Mile Logistics. Measuring and Computing Devices in Technological Processes. 2025. Vol. 84 (4). pp. 346-350. https://doi.org/10.31891/2219-9365-2025-84-41.
12. Юхимчук М.С., Лесько В.О., Дубовой В.М., Іванов Ю.Ю. Інтелектуальна система автоматичного керування процесом сушіння зернових культур на основі IoT-технологій. Наукові праці ВНТУ. Вінниця: ВНТУ, 2025. №4. С. 1-8. https://doi.org/10.31649/2307-5376-2025-4-46-53.

Панасюк Владислав Вадимович — студент групи 2ПКТ-24б, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, Вінниця, e-mail: bardivlad1@gmail.com

Науковий керівник: **Юхимчук Марія Сергіївна** — д-р техн. наук, професор кафедри комп'ютерних систем управління, Вінницький національний технічний університет, e-mail: umcmasha@gmail.com.

Лесько Владислав Олександрович — канд. техн. наук, доцент кафедри електричних станцій і систем, Вінницький національний технічний університет, e-mail: leskovlad@ukr.net.

Vladyslav Vadymovych Panasiuk – student of group 2PKT-24b, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: bardivlad1@gmail.com.

Supervisor: **Mariia Serhiivna Yukhymchuk** – Doctor of Technical Sciences, Professor of the Department of Computer Control Systems, Vinnytsia National Technical University, e-mail: umcmasha@gmail.com.

Vladyslav Oleksandrovych Lesko – Ph.D. in Technical Sciences, Associate Professor of the Department of Electric Stations and Systems, Vinnytsia National Technical University, e-mail: leskovlad@ukr.net.